APSI-BASED DELIBERATION IN GOAL ORIENTED AUTONOMOUS CONTROLLERS

Simone Fratini[1], Amedeo Cesta[1], Riccardo De Benedictis[1], Andrea Orlandini[1,2], and Riccardo Rasconi[1]

[1]*CNR, Consiglio Nazionale delle Ricerche, ISTC, Rome, Italy – name.surname@istc.cnr.it*
[2]*CNR, Consiglio Nazionale delle Ricerche, ITIA, Milan, Italy – andrea.orlandini@itia.cnr.it*

ABSTRACT

This paper describes a timeline-based, domain independent deliberative layer, based on ESA APSI technology, deployed in the context of the Goal Oriented Autonomous Controller (GOAC) project. In particular the paper describes a new controller composed by (1) a planning module that exploits the timeline-based approach provided by the APSI-TRF and is able to model and solve planning problems, (2) a module that dispatches planned timelines, supervises their execution status and entails continuous planning and re-planning. An example will illustrate both modules at work.

Key words: Timeline-based Planning, Planning and Execution, Autonomy.

1. INTRODUCTION

The Goal Oriented Autonomous Controller (GOAC [2]) is an ESA initiative aimed at defining a new generation of software autonomous controllers to support increasing levels of autonomy for robotic task achievement. In particular, goal of the GOAC architecture is to: (1) generate on-board plans, (2) dispatch activities for execution, and (3) recover from off-nominal conditions.

The pursued GOAC solution has been designed as a principled integration of different software solutions: (1) a timeline-based deliberative layer powered by a planner implemented on top of the APSI Timeline-based Representation Framework (APSI-TRF) created by ISTC-CNR, (2) a module, also based on APSI-TRF, to entail and supervise execution of plans and schedules, (3) an executive layer based on T-REX [6] devised at MBARI, and (4) a functional layer which integrates the G^{en}oM and BIP systems devised at LAAS and VERIMAG, respectively [1].

This work describes the CNR contribution in building a timeline-based, domain independent deliberative layer for the GOAC project. In particular we describe a controller entirely based on the ESA APSI technology. The controller is constituted by a new planning module that exploits the timeline-based approach provided by the APSI-TRF to model and solve problems, namely the "GOAC-APSI planner", as well as a module, namely the "APSI Deliberative Reactor"[1], to dispatch planned timelines, to supervise their execution status and to entail continuous planning and re-planning.

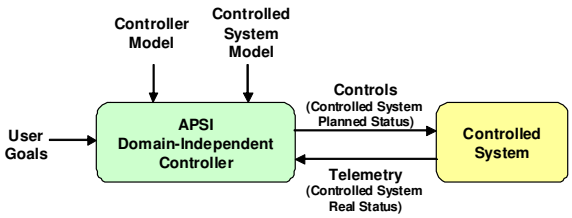

Figure 1. General Schema.

The controller is devised to be domain independent, i.e., it takes in input a (timeline-based) model of the system to be controlled, a (timeline-based) model of the specific controller and a set of user goal to be (hopefully) achieved by the controlled system (see Figure 1). The controller plans for user goals, dispatches commands for the controlled system (the status planned for the controlled system in order to achieve the user goals) and supervises plan execution by ingesting the telemetry of the controlled system (the actual status of the controlled system). The advantage of the controller being based on a domain independent planner entails both the capability of the controller to plan for user goal and to dynamically react to off-nominal conditions detected from the controlled system telemetry, as well as a twofold flexibility: in being used to control different systems (by substituting the controlled system domain description) and in the capability of achieving different classes of user goals for the same system (by substituting the controller model).

It is worth point out that the approach is scalable. In fact, the controlled system can be in turn another deliberative reactor and a hierarchical structure can be implemented to refine high levels goals into more detailed low level plans. Controls coming from the higher level reactors are in this case goals for the lower level reactors, and telemetry from lower levels reactors are plan achievement information for higher levels reactors. In the case of the GOAC project, the executive levels based on the T-REX architecture uses the reactors in such a hierarchical configuration (see [2] for more details).

[1]The term *Reactor* is a legacy from T-REX [6]. It is worth saying that the initial motivation of our work is to design a smooth integration with the T-REX executive that in its original implementation uses a different timeline-based planner.

2. THE APSI SOFTWARE PLATFORM AND THE GOAC-APSI PLANNER

Design and implementation of advanced Planning and Scheduling software for space applications is an activity involving a certain amount of developing effort and risk. For instance, the software may fail to meet operational requirements (performance issues), and/or may fail to capture all the essential aspects of the problem (modeling issues). The ESA Advanced Planning and Scheduling Initiative aims at the design and deployment of a software platform, called APSI Timeline Representation Framework, shortly APSI-TRF, for supporting planning and scheduling space applications design. The aim of the framework is to provide help to developers to cope with both software deployment efforts and modeling risks. The APSI-TRF simplifies the design and developing effort by providing a library of basic planning and scheduling domain independent solvers, and robustifies the interaction among the specific solvers implemented on top of the framework by providing a uniform representation of the solution database and defining a common inter-module cooperation and coordination interface. Modeling risks are reduced because the use of a framework that standardizes and simplifies the process of application deployment fosters a rapid prototyping cycle, which directly helps the users to take into account their own feedback during the application design.

The philosophy underlying the APSI-TRF is inspired by classical Control Theory, in that the problem is modeled by identifying a set of *relevant features* whose temporal evolutions need to be controlled to obtain a desired behavior. In the APSI-TRF such relevant features are called *components* and are the primitive entities for knowledge modeling. They represent logical or physical subsystems whose properties may vary in time; therefore, control decisions can be taken on components to define their evolution.

The APSI-TRF allows representing a number of planning and scheduling concepts in the form of timelines. The current APSI-TRF release provides two types of components which allow quite an amount of modeling power. Specifically, planning problems are modeled using components known as multi-valued state variables [5], coupled with resource components like those commonly used in constraint-based schedulers [3]. Besides problem solving capabilities, the APSI-TRF also provides a domain definition language (called DDL.3) to specify both the components and the relevant physical constraints that influence their possible temporal evolutions (e.g., possible state transitions over time of a component, synchronization/coordination constraints among different components, maximum capacity of resources, etc.), as well as a problem definition language (called PDL) to specify planning and scheduling problems. Section 4 shows an example of modeling with timelines.

The overall architecture of the interactions between the APSI-TRF, the GOAC-APSI planner and the GOAC Deliberative Reactor is shown in figure 2. The architecture is made of three main modules: (1) The APSI-TRF module is the P&S solution maintenance database. In this module the current solution is represented and maintained in the form of a network of constraints among planning and scheduling decisions. Modules built on top of the framework interact with the database by posting decisions and constraints and by querying for temporal and parameter information maintained in the database; (2) The GOAC-APSI planner, that interacts with the APSI-TRF to solve planning problems. The planner is used (in a master/slave configuration) by the deliberative reactor that provides problems and collect solutions (plans compliant with a domain description represented in DDL.3 language); (3) The APSI deliberative reactor, that interacts with the APSI-TRF to supervise the execution of the plans. The reactor is in charge of three tasks: enforcing the synchronization of the solution database with observations in input, providing plans for user goals and dispatching commands accordingly with its internal status. The reactor is described in Section 3.

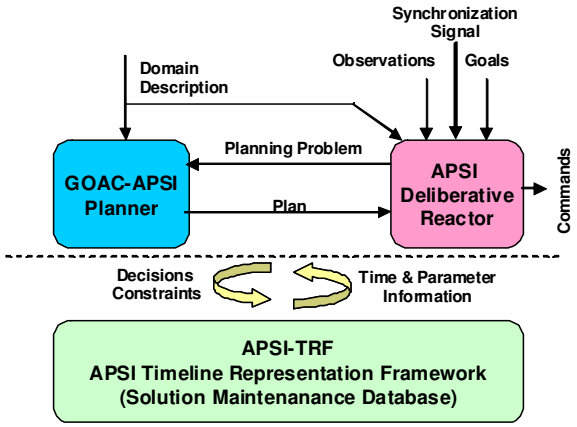

Figure 2. Architecture.

The GOAC-APSI planner is an enhancement/adaptation for GOAC purposes of the OMPS planner (Open Multi-Component Planner & Scheduler [4]) implemented on top of the APSI-TRF. The OMPS planner generates *time flexible* plans: a set of timelines where the transitions between the values are not fixed by the planner but temporally related with a set of constraints. Each transition has a minimum and a maximum planned time (in between of which the transition is legal and compliant with the model) as well as temporal relations with other transitions on the same timeline and on other timelines. These planned relations must be achieved to guarantee the validity of the plan. Any transition out of planned bounds or violating the relations is not valid and a plan execution failure must be taken into account.

The enhancement of the GOAC-APSI planner with respect to the original OMPS planner aim at enabling continuous planning in a dynamically evolving solution database. In fact, in GOAC, the solution database is concurrently (and continuously) modified by the reactor to synchronize the status of the database with the status of the real world, and by the planner to plan the (foreseen) status of the world in the future.

The adaptation consists in two main features: the capability to plan within a bounded time window and the possibility of being not blocking for the reactor. The first feature is used by the reactor to maintain the stability of the part of the plan currently in execution, while the second feature is necessary because the current implementation

of the APSI-TRF does not allow multi-thread concurrent access to the solution database. The GOAC-APSI planner implements a solving process as a sequence of short reasoning bursts (a few milliseconds each), so that it can be interrupted with no side effects on the portion of the timelines in execution (should synchronization problems occur that suddenly require the planner for more urgent duties). These reasoning steps allow to use the planner without interrupting concurrent reactor operations (see the reactor description for more details).

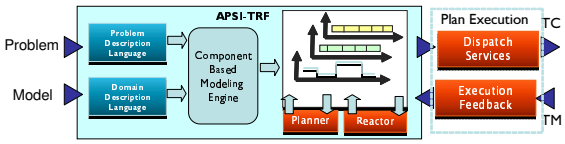

Figure 3. APSI-TRF *use in* GOAC.

The overall use of the APSI-TRF in the GOAC project is shown in Figure 3. The APSI-TRF provides: (1) languages for specifying domains and problems, (2) a unified plan and schedule representation database, and (3) two domain independent problem solvers: a planner and a reactor. Once connected with an infrastructure that allows to dispatch telecommands to the controlled system and ingest back telemetries (role plaid by T-REX in GOAC), the APSI-TRF plays the role of a timeline-based controller for the physical system.

3. THE APSI DELIBERATIVE REACTOR

The APSI deliberative reactor is the module responsible for supervising the plan generation and execution. The reactor implements a *sense-plan-act* cycle: (1) *sensing*, to enforce the synchronization of the believed status of the world (represented in the solution database) with the real status of the world (received as a series of *observations*); (2) *planning*, to collect *user goals* and synthesize action sequences that reach those goals consistently with the previous observations, and (3) *acting*, by dispatching *commands* accordingly with the reactor's internal status. The sense-plan-act cycle is synchronized with the real world by means of a *synchronization pulse* that triggers the cycle (see figure 2).

The APSI deliberative reactor design is grounded on two basic principles inherited from the philosophy of the T-REX executive architecture: (1) the scope of deliberation and execution are *temporally* partitioned; (2) the timelines that constitute the domain are *functionally* partitioned into *internal* and *external*.

The temporal separation between the scope of the deliberation and the scope of the execution aims at providing sufficient time for the planning phase before the produced plan execution so as to avoid any overlap between the timeline segments being planned and those under execution, guaranteeing the stability of the controller. This principle implies the definition of a *latency* λ and a *deliberation scope* π for the reactor. Latency and deliberation scope separate the portion of the timeline subject to planning with the portion of the timeline subject to execution. During execution, a planning process starting at a given time t, is guaranteed to modify the timelines only between $t + \lambda$ and $t + \lambda + \pi$, therefore maintaining timeline stability between t and $t + \lambda$. λ can be considered an empirical evaluation of the maximum allotted time for planning for a goal, while π can be considered an empirical evaluation of the temporal scope of a plan. Hence the importance of accurately choosing the values of λ and π depending on (1) the presumed performances of the planner in use, (2) the complexity of the domain theory describing the system to be controlled, and (3) the difficulty of the goals to be dynamically achieved during the execution to control the system.

The functional distinction between internal and external timelines aims at distinguishing between the timelines that describe the controller internal status and the timelines that describe the status of the system that has to be controlled (basically the distinction between the controller model and the controlled system model in Figure 1). This distinction is important, as the internal status of the controller can be decided by the reactor (i.e., it will surely be reached as expected), while the status of the controlled system can only be planned for by the reactor but there is no guarantee that it will be actually reached.

The reactor receives the current status of the controlled system through a series of telemetry values (i.e., actual values of external timelines, a.k.a. *observations*), and attempts to synchronize its own view of these timelines in the plan database (believed status of the controlled system) with the *observed* timeline values. Interleaved with this activity, the reactor plans[2] for the desired status of the controlled system required to achieve user goals, and dispatches this status by means of *controls*, i.e., required values for external timelines. The actual achievement of these values is supervised by means of the synchronization process described below.

Three main modules are in charge of the reactor functions (see Figure 5): a *core*, that supervises the process, an *observer*, that synchronizes incoming observations and a *dispatcher*, that dispatches outcoming commands. A fourth module, a *planner*, is needed to plan for user goals[3].

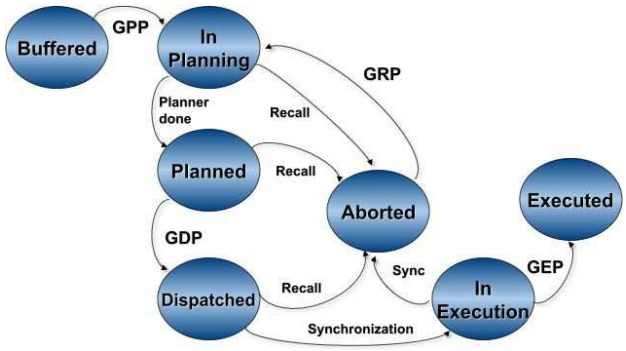

Figure 4. User Goal Life Cycle.

The core module coordinates the other modules while

[2] According to the limitations imposed by the λ and π values

[3] In the GOAC project the planner in use is the GOAC-APSI planner described in Section 2, but this module can in principle use any APSI-based solver

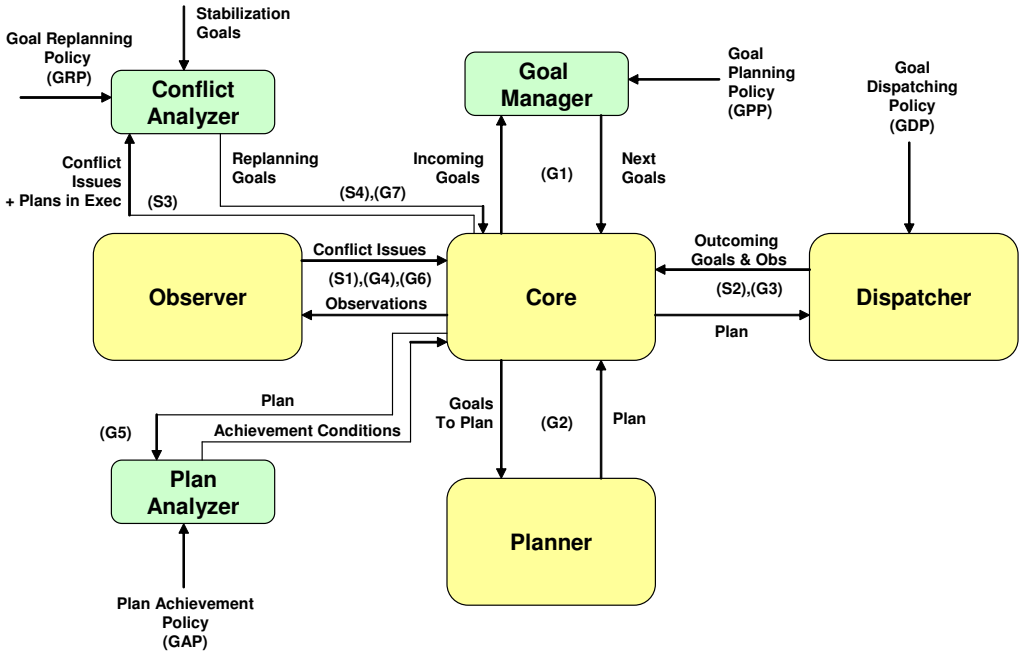

Figure 5. Reactor Architecture

managing planning processes and synchronization procedures. The core module implements the user goal life cycle shown in Figure 4. With reference to the labels in Figure 5, the goal life cycle is implemented according to the following steps: (G1) incoming user goals are collected and buffered into a *goal manager* module (the goal is BUFFERED); a "Goal Planning Policy" (GPP) implemented in the goal manager module controls the progression of the goal from the status BUFFERED to the status IN PLANNING. (G2) the goals selected by the GPP to be planned next are passed on to the planner. Once the time allotted for planning has expired, the transition from IN PLANNING to PLANNED is triggered. (G3) Once planned, the produced plan is passed on to the Dispatcher module. A "Goal Dispatching Policy" (GDP) implemented by the dispatcher controls the progression of the goal from PLANNED to DISPATCHED. Once selected to be dispatched by the GDP, the goal is sent to the controlled system. (G4) the goals come in execution as soon as any of the values that constitute the plan gets in execution. The Observer module, in charge of supervising the execution of the reactor timelines, triggers the transition from DISPATCHED to IN EXECUTION. (G5) once in execution, a "Goal Achievement Policy" (GAP) controls the progression from IN EXECUTION to EXECUTED. A *plan analyzer* module implements the GAP. The module analyzes the plan and produces a set of *goal achievement conditions* that the *core* uses to assess the correct execution of the plan. (G6) The Observer module, while synchronizing timelines, can force the transition of the goal from the status IN EXECUTION to ABORTED as soon as any synchronization problem arises (an incorrect execution of any of the values of the plan is detected). In such case, (G7) a "Goal Re-Planning Policy" (GRP) controls the possible transition of the goal status from ABORTED to IN PLANNING again. A *conflict analyzer* modules implements the GRP. The GRP evaluates the situation on the basis of a set of *conflict issues* generated by the Observer module, and can decide if and when to re-plan for the goal. When the goal is BUFFERED, IN PLANNING, PLANNED and DISPATCHED, it can be *recalled* (by the user) forcing a transition into a ABORTED status. The current implementation of the reactor does not allow to recall a goal once the plan for that goal is in execution.

The goal life cycle management process is interleaved with the timeline synchronization process. Again with reference to the labels in figure 5, the following steps constitute the reactor synchronization procedure: (S1) incoming observations that describe the actual status of the controlled system are passed on to the Observer module. (S2) if no synchronization problem arises, the observed status of the controlled system is compatible with the current plan database; the plan database is updated and the out coming controls are produced. (S3) if a synchronization problem arises, the current status of both the controller and controlled system is re-calculated, and a list of *conflict issues* is generated for the *Conflict Analyzer*. (S4) On the basis of the implemented GRP, the module provides the core with a set of goals to re-plan for (starting from the re-calculated status). The GRP can decide to either (a) re-plan from the goal former in execution or any other goal still pending, or (b) plan for a set of pre-defined "stabilization goals", a sort of standard "safe" status that the controller will try to reach before attempting to plan for new goals or re-plan for the aborted goals. Conflicts are detected by the observer module when (1) an incoming observation puts an unexpected value on a timeline forcing an invalid state transition (the controlled system is taking an unexpected status), (2) an incoming observation forces a transition on a state that cannot be changed (the controlled system truncates an operation before the planned time) or (3) when there is no incoming observation but the current status cannot be maintained (the controlled system is not blocking an operation at the planned time).

The reactor is designed to be modular and easily extensible with respect to the implemented policies. This features makes the behavior of the reactor easily customizable for different domains. In fact, especially for what the goal goal planning and re-planning policy (GPP and GRP) are concerned, it is very difficult to design scenario independent policies that are general and efficient at the same time. Moreover the policies can change also for different runs of the same scenario, to achieve different user needs. Hence a design choice of bet on modularity more than implementing embedded policies. The next section shows an illustrative scenario with an example of problem dependent policies.

4. AN ILLUSTRATIVE SCENARIO

This section describes an experiment done at LAAS in the context of the GOAC project. The goal of the experiment was to control a rover to take a set of pictures, store them on board and dump the pictures when a given communication channel was available. The target platform was DALA, a mature platform that has been successfully used for several years. DALA robot is one of the LAAS robotic platforms that can be used for autonomous exploration type of experiments. DALA is an iRobot ATRV robot that provides a large number of sensors and effectors. Nevertheless, it can use vision based navigation (such as the one used on Spirit and Opportunity), as well as indoor navigation based on a Sick laser range finder. The experiment has been performed using the T-REX-based executive layer and a functional layer for connecting to the DALA rover (which integrates the G^{en}oM and BIP systems).

The first step was to model the system to be controlled: the rover. The DALA rover is equipped with a pan-tilt unit, two stereo cameras (mounted on top of the pan-tilt unit) and a communication facility. The rover is able to autonomously navigate the environment, move the pan-tilt, take pictures and communicate images to a remote orbiter. Hence we consider, for this experiment, as interesting sub-systems to model the DALA rover: a mobility system MS, a pan-tilt unit PTU, a camera CAM and a communication system COMM.[4]

We consider in the model the rover able to move between two points in space given their coordinates $\langle x, y\rangle$, taking into account that the rover may get stuck in between two points. Hence the mobility system can be modeled as a state variable taking the following values: AT$(?x, ?y)$ when the rover is standing in $\langle x, y\rangle$, GOTO$(?x, ?y)$ when the rover is moving toward $\langle x, y\rangle$ and STUCKAT$(?x, ?y)$ when the rover is stuck in $\langle x, y\rangle$. We assume that a transition GOTO$(?x, ?y) \rightarrow$ AT$(?x, ?y)$ denotes a successful move to $\langle x, y\rangle$, while a transition GOTO$(?x, ?y) \rightarrow$ STUCKAT$(?x', ?y')$, with $?x \neq ?x'$ or $?y \neq ?y'$ denotes an unsuccessful move to $\langle x, y\rangle$ with the rover stuck in $\langle x', y'\rangle$ while moving. A transition AT$(?x, ?y) \rightarrow$ GOTO$(?x', ?y')$ denotes the rover starting to move from a point $\langle x, y\rangle$ to a point $\langle x', y'\rangle$.

[4]We are indebted to Felix Ingrand and Lavindra De Silva from LAAS-CNRS for the time spent to explain us the details of their robotic platform.

The rover pan-tilt unit can be pointing a given angle $\langle \alpha, \beta\rangle$ or can be moving from two angles. Hence the unit can be modeled with a state variable taking the following values: POINTINGAT$(?pan, ?tilt)$ when the unit is is pointing an angle $\langle \alpha = ?pan, \beta = ?tilt\rangle$ and MOVINGTO$(?pan, ?tilt)$ when the unit is moving toward an angle $\langle \alpha = ?pan, \beta = ?tilt\rangle$.

The rover camera can take pictures with the pan-tilt unit (in the current position of the rover) and store the picture on an on-board memory with a given file id. Hence the camera can be modeled with a state variable taking the following values: CAMIDLE(), when the camera is not taking pictures and TAKEPIC$(?file_id)$ when the unit is taking a picture that will be stored in a file with id = $?file\_id$.

The communication system can dump a file with a given id. Hence it can be modeled with a state variable taking the following values: COMMIDLE(), the idle status, and DUMP$(?file_id)$ when the unit is dumping a picture stored in a file with id = $?file\_id$.

There are some constraints that have to be satisfied in order to correctly use the rover. The first one (C1) is that when the rover is taking a picture, it must be stable at a given location. The second (C2) is that when the rover is moving, the pan-tilt unit must be in a "rest" position (i.e., at angle $\langle 0, 0\rangle$). We model these requirements stating the following constraints:

CAM.TAKEPIC$(?file_id)$ DURING MS.AT$(?x, ?y)$ (C1)
MS.GOTO$(?x, ?y)$ DURING PTU.POINTINGAT$(0, 0)$ (C2)

The second step was to model the controller to operate the DALA rover. Interesting user goals were: (1) to take a picture in a given position $\langle x, y\rangle$ with the pan-tilt unit in $\langle \alpha, \beta\rangle$, storing it with a given file id and dump it when the orbiter is visible for some periods and (2) to drive the rover in a given position $\langle x, y\rangle$. Hence we modeled the controller using two timelines: a mission timeline MT and a visibility dump window timeline VW.

The mission timeline will take the values TAKEPICTURE$(?x, ?y, ?pan, ?tilt, ?file_id)$, to model the goal of taking a picture in $\langle ?x, ?y\rangle$, with the pan-tilt unit pointing to $\langle ?pan, ?tilt\rangle$, store the picture in a file with the id $?file\_id$ and dump it as soon as possible, the value GOTO$(?x, ?y)$ to model the goal of just moving the rover to $\langle ?x, ?y\rangle$ and IDLE() (an idle status).

The visibility window timeline models the temporal windows where the communication channel is available. For the sake of simplicity, we model a binary state variable with only two values: AVAILABLE() and NOTAVAILABLE().

The goal TAKEPICTURE$(?x, ?y, ?pan, ?tilt, ?file_id)$ can be achieved by the rover by: (a) taking a picture with id = $?file\_id$, with the rover in $\langle ?x, ?y\rangle$ and the pan-tilt unit pointing to $\langle ?pan, ?tilt\rangle$ and (b) dumping the picture. To take a picture on a given position and pan-tilt unit orientation, can be achieved by (a.1) moving the rover to $\langle ?x, ?y\rangle$, (a.2) moving the pan tilt unit to $\langle ?pan, ?tilt\rangle$ (a.3) taking a picture with the camera with an id = $?file\_id$. To dump a picture with a given id can be achieved by (b.1) dumping the picture when the dump

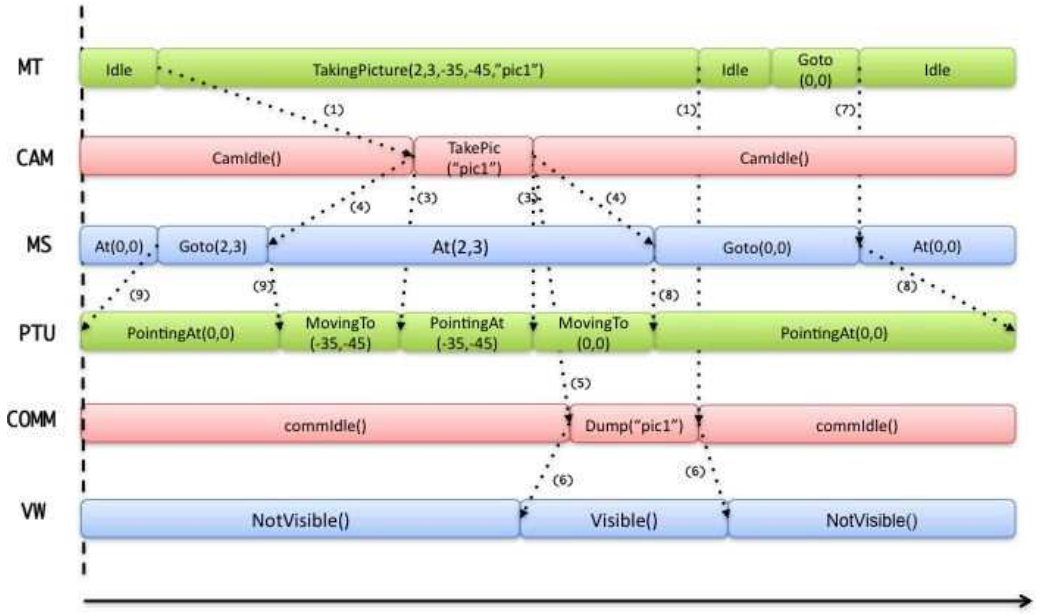

Figure 6. A Time Flexible Plan

window is available. Hence we add to the model the following synchronization:

MT.TAKEPICTURE($?x, ?y, ?pan, ?tilt, ?file\_id$)
CONTAINS
⟨ CAM.TAKEPIC($?file\_id$) BEFORE COMM.DUMP($?file\_id$),
CAM.TAKEPIC($?file\_id$) DURING MS.AT($?x, ?y$),
CAM.TAKEPIC($?file\_id$) DURING PTU.POINTINGAT($?pan, ?tilt$),
COMM.DUMP($?file\_id$) DURING VW.AVAILABLE() ⟩

The goal GOTO($?x, ?y$) can be achieved by driving the rover to $\langle ?x, ?y \rangle$. Hence we need to add to the model just the following synchronization:

MT.GOTO($?x, ?y$) MEETS MS.AT($?x, ?y$)

Figure 6 shows an example of a plan to take a pictures and to go back to the rest position. We have two user goals in the Mission Timeline MT: TAKEPICTURE($2, 3, -35, -45, '' pic1''$) and GOTO$(0, 0)$. The initial values of the timelines are provided to the controller for internal timelines (MT and VW), while the initial values of external timelines (that describes the initial status of the rover) are ingested as observation after the first synchronization pulse. We have an idle status for the mission timeline and a complete description of the visibility windows for what internal timelines are concerned. The rover takes the status with camera and communication in idle, position in $\langle 0, 0 \rangle$ and pan-tilt unit in $\langle 0, 0 \rangle$.

The plan for the two goals is shown in Figure 6,where we have also an example of the temporal flexibility property introduced in Section 2. With reference to the labels, we have the MT.TAKEPICTURE($2, 3, -35, -45, '' pic1''$) goal containing CAM.TAKEPIC($''pic1''$) and COMM.DUMP($''pic1''$) (the two constraints labeled with (1)), with CAM.TAKEPIC($''pic1''$) before COMM.DUMP($''pic1''$) (the constraint labeled with (5)), while CAM.TAKEPIC($''pic1''$) is placed during MS.AT$(2, 3)$ (constraints (4)) and during PTU.POINTINGAT$(-35, -45)$ (constraints (3)). The dump operation is placed during the visibility window (constraints (6)). All the commands that move the rover are placed when the pan tilt unit is in the default position (constraints (8) and (9)). This comes from the model of the DALA rover. Finally, the goal GOTO$(0, 0)$ meets a status MS.AT$(0, 0)$ that achieves the goal (constraint (7)).

The transitions between the different planned timeline values are temporally related but not fixed by the planner. The relations labeled in the figure must be achieved to guarantee the validity of the plan. Any transition out of planned bounds or violating the relations is not valid and a failure is detected by the reactor.

A violation is detected also if the controlled system takes an unexpected status, like STUCKAT$(?x, ?y)$ in the case of this illustrative scenario. At this point, the goal in execution (let's suppose the unexpected event happening while executing a plan to take a picture) will be considered discarded by the reactor, the current status of the plan database is updated to be compliant with the status of the controlled system and the goal re-planning policy selects the new goal to plan for. The policy implemented for this scenario considers the position where the rover is stuck and re-schedule the goals on the basis of the distance where the rover is stuck. This is an example of a very simple goal re-planning policy related to the specific scenario. Different more sophisticated policies can be implemented and easily added to reactor in order to achieve a more realistic behavior.

5. CONCLUSIONS

This paper describes a timeline-based, domain independent deliberative layer based on ESA APSI technology built for the Goal Oriented Autonomous Controller (GOAC) project.

The controller is designed to be domain independent, i.e., it takes in input a (timeline-based) model of the system to be controlled, a (timeline-based) model of the specific controller and a set of user goal to be (hopefully) achieved

by the controlled system. The controller uses a planner to plans for user goals, dispatches commands for the controlled system (the status planned for the controlled system in order to achieve the user goals) and supervises plan execution by reasoning on the telemetry of the controlled system (the actual status of the controlled system).

The proposed controller is constituted by:

- a new planning module, referred to as the "GOAC-APSI planner" that exploits the timeline-based approach provided by the APSI-TRF to model and solve timeline-based planning problems. The GOAC-APSI planner generates time flexible plans: a set of timelines where the transitions between the values are not fixed by the planner but temporally related with a set of constraints. The flexibility of the produced plans and the capability of re-planning entail the suitability of the system in controlling physical systems, once coupled with an executive layer that takes care of properly dispatching the plan and supervise its execution.
- a module, called the "APSI Deliberative Reactor", that implements the *sense-plan-act* cycle as well as the goal life-cycle management. The module dispatches planned timelines and supervises their execution status to enforce continuous synchronization with the status of the controlled system and to allow planning and re-planning.

The advantage of the controller being based on a domain independent planner entails both the capability of the controller to plan for goal and to dynamically react to off-nominal conditions detected from the controlled system telemetry, as well as a twofold flexibility: in being used to control different systems (by substituting the controlled system domain description) and in the capability of achieving different classes of user goals for the same system (by substituting the controller model).

Acknowledgment. Authors are partially supported by the European Space Agency (ESA) under the GOAC project (TRP/T313/006MM). We gratefully thanks our colleagues in the GOAC project for the fruitful discussions. Orlandini is currently supported by a grant within "Accordo di Programma Quadro CNR-Regione Lombardia: Progetto 3".

REFERENCES

1. S. Bensalem, L. de Silva, M. Gallien, F. Ingrand, and R. Yan. "Rock Solid" Software: A Verifiable and Correct-by-Construction Controller for Rover and Spacecraft Functional Levels. In *i-SAIRAS-10. Proc. of the 10^{th} Int. Symp. on Artificial Intelligence, Robotics and Automation in Space*, 2010.

2. A. Ceballos, S. Bensalem, A. Cesta, L. de Silva, S. Fratini, F. Ingrand, J. Ocón, A. Orlandini, F. Py, K. Rajan, R. Rasconi, and M. van Winnendael. A Goal-Oriented Autonomous Controller for space exploration. In *Proceedings of the ASTRA 2011, 11^{th} Symposium on Advanced Space Technologies in Robotics and Automation*, 2011.

3. A. Cesta, A. Oddi, and S. F. Smith. A Constraint-based method for Project Scheduling with Time Windows. *Journal of Heuristics*, 8(1):109–136, January 2002.

4. S. Fratini, F. Pecora, and A. Cesta. Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. *Archives of Control Sciences*, 18(2):231–271, 2008.

5. N. Muscettola. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., editor, *Intelligent Scheduling*. Morgan Kauffmann, 1994.

6. F. Py, K. Rajan, and C. McGann. A systematic agent framework for situated autonomous systems. In *AAMAS*, pages 583–590, 2010.